Claims 2-22 are pending in the present application. Reconsideration of the claims is respectfully requested.

I. 35 U.S.C. § 101

The Examiner has rejected Claim 22 under 35 U.S.C. § 101. This rejection is respectfully traversed.

Applicants consider that Claim 22 is directed to statutory subject matter for the reasons set forth in their Amendment Pursuant to a Request for Continued Examination, filed with the USPTO on December 26, 2007. As stated therein, Claim 22 is considered to be statutory subject matter in view of provisions of **MPEP 2106**. Regarding non-statutory subject matter, the **MPEP** states:

> In this context, "functional descriptive material" consists of **data structures** and computer programs **which impart functionality when employed as a computer component**. (The definition of "data structure" is "a physical or logical relationship among data elements, designed to support specific data manipulation functions." The New IEEE Standard Dictionary of Electrical and Electronics Terms 308 (5th ed. 1993).) "Nonfunctional descriptive material" includes but is not limited to music, literary works and a compilation or mere arrangement of data.
>
> **When functional descriptive material is recorded on some computer-readable medium it becomes structurally and functionally interrelated to the medium and will be statutory in most cases since use of technology permits the function of the descriptive material to be realized**. Compare *In re Lowry*, 32 F.3d 1579, 1583-84, 32 USPQ2d 1031, 1035 (Fed. Cir. 1994) (claim to data structure stored on a computer readable medium that increases computer efficiency held statutory) and *Warmerdam*, 33 F.3d at 1360-61, 31 USPQ2d at 1759 (claim to computer having a specific data structure stored in memory held statutory product-by-process claim) with *Warmerdam*, 33 F.3d at 1361, 31 USPQ2d at 1760 (claim to a data structure *per se* held nonstatutory). (**emphasis added**) **MPEP 2106 (IV)(B)(1)**

Claim 22 recites clearly functional descriptive material, since such material imparts functionality when employed as a computer component. Moreover, the functional descriptive material of Claim 22 is recorded on "some" computer readable medium.

In the above context, the term "some" means "any" computer readable medium. The MPEP does not draw any distinctions between one type of media that is considered to be statutory and another type of media that is considered to be non-statutory. To the contrary, the MPEP clearly states that as long as the functional descriptive material is in "some" computer readable medium, it should be considered statutory. The only exception to this statement in the MPEP is functional descriptive material that does not generate a useful, concrete and tangible result, e.g., functional descriptive material composed completely of pure mathematical concepts that provide no practical result. Claim 22 clearly recites a useful, concrete and tangible result by using a plug-in class loader that is associated with and delegates to an identified class loader in order to load a plug-in class that is associated with a specified application class.

Accordingly, Claim 22 is directed to functional descriptive material that provides a useful, concrete and tangible result, and which is embodied on "some" computer readable medium. Therefore, Claim 22 is statutory, and the rejection thereof under 35 U.S.C. § 101 has been overcome.

II. 35 U.S.C. § 103, Obviousness

The Examiner has rejected Claims 2-3, 7-12 and 17-22 under 35 U.S.C. § 103, as being unpatentable over U.S. Publication No. 2004/0015936 to *Susarla et al.* (hereinafter "*Susarla*"). The Examiner has rejected Claims 4-6 and 13-16 under 35 U.S.C. § 103 as being unpatentable over *Susarla* in view of teachings related to Figure 5B of the application, which Applicants provided to illustrate a problem of the prior art. These rejections are respectfully traversed.

III. Teachings of Applicants

In their invention, Applicants provide a plug-in loader for each existing application class loader associated with a class loader hierarchy. In addition, a different plug-in class loader is provided for each class loader in the class loader hierarchy, wherein each plug-in class loader is associated with only a single class loader of the hierarchy, and each plug-in class delegates to its associated class loader. The class loader used to load a specified application class is identified, and the plug-in class loader associated with the identified class loader is then used to load any plug-in class that is associated with the specified application class. Embodiments of the invention make it unnecessary to modify class path in order to load plug-in classes, and also provide other benefits and advantages as described in the application. Moreover, embodiments of the invention provide a generic mechanism for any application to load plug-in classes at an appropriate level in the class loader hierarchy.

These teachings are found in the specification such as at page 18, lines 25 - page 19, line 2; page 19, line 24 - page 20, line 4; page 20, lines 11-20; and at Figure 6B. Figure 6B is shown below:

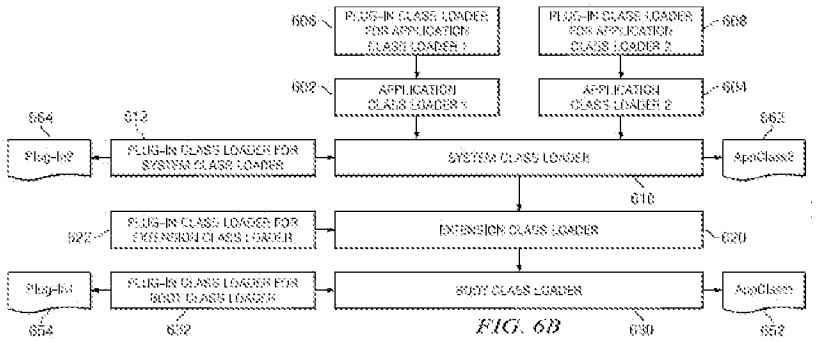

FIG. 6B

Claim 2 recites one useful embodiment of the invention. Applicants' Claim 2 now reads as follows:

> 2. A computer implemented method for selecting a class loader to load a plug-in class within a class loader heirarchy, the method comprising the steps of:
>
> generating a class loader hierarchy comprising a plurality of class loaders that includes two or more application class loaders, one for each of two or more application classes, wherein each of said application class loaders can selectively load its application class, or delegate the loading of its application class to another class loader of said class loader hierarchy;
>
> providing a different plug-in class loader for each class loader of said class loader hierarchy, wherein each plug-in class loader is associated with only a single class loader of said hierarchy, and each plug-in class loader delegates to its associated class loader;
>
> identifying the class loader of said hierarchy that is used to load a specified one of said two or more application classes; and
>
> using the plug-in class loader that is associated with and delegates to said identified class loader to load a given plug-in class that is associated with said specified application class.

The Examiner bears the burden of establishing a prima facie case of obviousness based on prior art when rejecting claims under 35 U.S.C. § 103. *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992). In order to establish a *prima facie* case of obviousness under 35 U.S.C. § 103, *Graham v. John Deere Co. of Kansas City*, 383 U.S. 1 (1966) requires determining, respectively, the scope and content of the prior art, the differences between the prior art and the claims at issue, and the level of ordinary skill in the pertinent art. Rejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning, to support the legal conclusion of obviousness. *KSR Int'l. Co. v. Teleflex, Inc.*, No. 04-1350 (U.S. Apr. 30, 2007) (citing *In re Kahn*, 441 F.3d 977, 988 (CA Fed. 2006)). Additionally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. *In re Royka*, 490 F.2d 981, 180 USPQ 580 (CCPA 1974). The suggestion to make the claim combination must be found in the prior art, not in the Applicants' disclosure. *In re Vaek*, 20 U.S.P.Q.2d 1438 (Fed. Cir. 1991). Moreover, in accordance with **MPEP § 2142.02**, each prior art reference must be considered in its entirety, i.e., as a whole, including portions that would lead away from the claimed invention. W.*L. Gore & Associates, Inc.* v. *Garlock, Inc.*, 220 U.S.P.Q. 303 (Fed. Cir. 1993). A third essential requirement for establishing a prima facie case, set forth in **MPEP § 2143.01**, is that the proposed modification cannot render the prior art unsatisfactory for its intended purpose.

In the present case, not all of the features of the claimed invention have been properly considered, and the teachings of the reference itself do not teach or suggest the claimed subject matter to a person of

ordinary skill in the art. For example, *Susarla* fails to teach or suggest, in the over-all combination of Claim 2, either of the following features:

(1) Providing a different plug-in class loader for each class loader of the class loader hierarchy, wherein each plug-in class loader is associated with only a single class loader of the hierarchy, and each plug-in class loader delegates to its associated class loader (hereinafter "Feature (1)").

(2) Using the plug-in class loader that is associated with and delegates to the identified class loader to load a plug-in class that is associated with the specified application class (hereinafter "Feature (2)").

IV. Feature (1) of Claim 2 Distinguishes over Susarla Reference

Feature (1) of Applicants' Claim 2 is directed to providing a different plug-in class loader for each class loader in a class loader hierarchy. Moreover, each plug-in class loader is associated with only a single class loader of the hierarchy, and each class loader delegates to its corresponding class loader. This feature is disclosed in Applicants' specification, such as at page 20, lines 11-22, taken together with Figure 6B of Applicants' drawings, and also at page 24, lines 3-9. Figure 6B shows a class loader hierarchy comprising the class loaders 610, 620 and 630, and further comprising class loader 602 and 604 for application classes 1 and 2, respectively. Figure 6B further shows each of the plug-in loaders 606, 608, 612, 622 and 632, wherein each class loader is provided with a different plug-in class loader, and each plug-in class loader is associated with only one of the class loaders 602-604 or 610-630. As taught in the specification at page 24, lines 6-9, Feature (1) of Claim 2 is essential in providing the benefit of a generic mechanism, for enabling any application to load plug-in classes at an appropriate level in the class loader hierarchy.

Pertinent teachings of Susarla include teachings found at the abstract thereof; at paragraphs [0017], [0056], [0059], [0101] and [0138]; and at Figure 4. Figure 4 of Susarla is as follows:

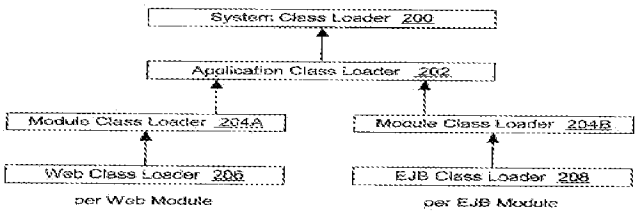

Fig. 4

In the Office Action, it is apparently asserted that the module class loaders 204A and 204B of Susarla are the components thereof that disclose or are equivalent to the respective plug-in class loaders recited by Feature (1) of Claim 2. For example, the following is stated in the Office Action:

> **Claim 2:**
>
> In the 2nd par. on pg. 13, the Applicants assert:
>
> Susarla, in fact, appears to provide no teaching whatever in regard to the loading of plug-ins, which are extensions meant to extend the behavior of application classes, as is known by those of ordinary skill in the art. To the contrary, Susarla is concerned with an entirely different issue, that is, the dynamic reloading of classes, which may become necessary when classes used by an application have been changed.
>
> The examiner respectfully disagrees. As indicated in the rejection Susarla's 'module class loaders' (e.g. Fig. 4, 204A-B) have been mapped to the claimed 'plug-in class loaders'. The Applicants have not asserted a functional distinction between these objects. Those of ordinary skill in the art would have recognized that the disclosed code 'modules' provide functionality to an application and thus 'extend the behavior' of that application. Further Susarla discloses "Each module in the application may be associated with its own class loader" (e.g. abstract). Accordingly, Susarla anticipates the claimed plug-in class loaders. **[Office Action, 2/7/2008, pp. 2-3]**
>
> providing a plug-in class loader for each application class loader in the class loader hierarchy (Fig 4, Module Class Loader 204A), wherein each plug-in class loader is associated with only a single class loader of said hierarchy, and each plug-in class loader delegates to its associated class loader (Fig. 4 Application Class Loader 202; note class loaders 206 and 208 are optional (see par. [0101]), thus loaders 204A and B are disclosed, in some embodiments, as associated with only a single class loader i.e. loader 202); **[Office Action, 2/7/2008, pp. 5-6]**

Accordingly, it is necessary to closely examine the module class loaders 204A and 204B, as well as their relationship with other components as disclosed by the *Susarla* arrangement. Such arrangement must then be compared with the recitation of Feature (1) of Claim 2. As a result of such examination and comparison, it becomes readily apparent that *Susarla* fails to disclose the recitation of Feature (1), for at least Reasons 1-4 set forth hereinafter.

Reason 1

Susarla such as at paragraph [0056] states explicitly that the class loader module thereof "may include a hierarchical stack of class loaders". The statement "zero or more child loaders" apparently indicates that a class loader module stack can comprise one or more class loaders. Also, the explicit teaching, that a class loader module 204A or 204B may include a hierarchical stack of class loaders, is further emphasized by *Susarla,* such as at the abstract thereof, and at paragraphs [0057] and [0138]. In view of these clear teachings, Figure 4 of *Susarla* including its modules 204A and B discloses only a number of class loaders, wherein each of the class loaders is included in a class loader hierarchy.

Since Figure 4 comprises a number of class loaders that are each arranged into a class loader hierarchy, each such class loader, including each of the class loaders of modules 204A and 204B, must have a different plug-in class loader provided for it, in order to fully disclose the recitation of Feature (1) of Claim 2. Moreover, each plug-in class loader must be associated with only a single class loader of the hierarchy. These requirements of Feature (1) are clearly shown by Figure 6 of Applicants' drawings, as discussed above. In order for the arrangement of Figure 4 of *Susarla* to disclose these requirements, there would have to be a different plug-in class loader for each class loader in the hierarchies of both modules 204A and 204B. There would also have to be a separate plug-in class loader for the system class loader 200, and different plug-in loaders for web class loader 206 and EJB class loader 208.

It is readily apparent, based on the clear and fair teaching of *Susarla* as shown by Figure 4, that *Susarla* neither discloses nor suggests any such provision or use of plug-in class loaders, as is taught by Feature (1) of Applicants' Claim 2.

Reason 2

Figure 4 clearly shows two module class loaders 204A and 204B, both linked to and thus associated with the same application class loader 202, wherein class loader 202 is further shown to be part of a class loader hierarchy. If each module class loader 204A and 204B is in fact equivalent to a plug-in class loader as recited by Feature (1) of Claim 2, then *Susarla* clearly teaches away from the Feature (1) requirement that each plug-in class loader is associated with only a single class loader of a hierarchy.

Reason 3

If a module class loader 204A or 204B was equivalent to the plug-in class loaders of Feature (1) of Claim 2, as asserted by the Examiner, then a different one of such modules would have to be provided for each class loader in the hierarchy of Figure 4 of *Susarla*, in order to disclose the recitation of Applicants' Feature (1) in its entirety. For example, an additional module class loader, which was like one of the modules 204A or B, would have to be shown in association with system class loader 200 of Figure 4. However, it is readily apparent that no such teaching is shown or suggested, either by Figure 4 of *Susarla* or by any other section thereof.

Reason 4

A plug-in is a program that typically provides a specific function on demand. Applicants use plug-ins to extend the behavior of application processes. *Susarla*, however, is concerned with dynamic reloading of application classes and not with the behavior thereof, as is further emphasized such as at paragraph [0138] and Figure 7 thereof. *Susarla* therefore has no need for plug-ins, and neither discloses

nor makes any reference thereto. Accordingly, *Susarla* neither needs nor discloses plug-in loaders, as recited by Feature (1) of Claim 2.

V. Feature (2) of Claim 2 Distinguishes Over Susarla Reference

Feature (2) of Claim 2 recites using the plug-in class loader that is associated with and delegates to the identified class loader to load the plug-in class that is associated with the specified application class. *Susarla*, of course, cannot show this feature, since it fails to disclose the plug-in class loader required for Feature (2), as was discussed above in connection with Feature (1).

Moreover, paragraph [0138] of *Susarla*, cited in the Office Action in connection with Feature (2), once again discloses a class loader module that includes a hierarchical stack of class loaders, for dynamic class reloading as discussed above. Paragraph [0138] further teaches that a class loader controller can be configured to receive a request to load a class, and in response to receiving a request, may locate the appropriate class loader in the stack, and then invoke it. Clearly, this teaching of Susarla fails to show or suggest using a plug-in that is associated with and delegates to an identified class loader, in order to load a plug-in class, as recited by Feature (2) of Claim 2.

VI. Susarla Does Not Support 35 U.S.C. § 103 Rejection of Claim 2

In the Office Action, the Examiner acknowledged that Susarla fails to show the Feature (1) recitation of providing a different plug-in class loader for each class loader in the hierarchy. Therefore, to support a rejection under 35 U.S.C. § 103, the Examiner stated the following:

> 10. It would have been obvious to one of ordinary skill in the art at the time the invention was made to provide a different plug-in class loader (Fig. 4, Module Class Loader 204A) for each class loader in the Hierarchy (i.e. Fig. 4, System Class Loader 200) in order to provide 'dynamic class reloading' (see Abstract) at the system level as in addition to the application level (par. [0007] "the system class loader loads the standard classes and the application server core classes, and the application class loader loads the user-defined classes"). **[Office Action, 2/7/2008, p. 6]**

No support is provided for the above statement, except for references to the abstract and paragraph [0007] of *Susarla*. However, each of these sections teaches that a separate class loader or class loader module is provided only for each application. Thus, neither section discloses or suggests providing a separate class loader module for the system class loader. In fact, paragraph [0007] of *Susarla* states that the system class loader is the parent of the application class loaders. Paragraph [0007] thereby teaches that a separate class loader module for the system class loader would be unnecessary, since reloading of any application class would automatically be directed to its parent system class loader, as shown by Figure 4 of *Susarla*. Accordingly, those of skill in the art are directed away from providing a separate class loader module for the system class loader of *Susarla*, since such separate module would incur an

additional resource cost. Those of skill in the art would not want to increase the cost by providing a component that was not necessary. Thus, *Susarla* teaches away from an essential element of Applicants' Claim 2, so that Claim 2 is clearly non-obvious in view of *Susarla*.

Moreover, as discussed above, *KSR Int'l Co. v. Teleflex, Inc.* (supra) holds that there must be some articulated reasoning with some rational underpinning, in order to support a legal conclusion of obviousness. Applicants consider that such requisite articulated reasoning has not been provided.

VII. Claim 3 Distinguishes over Cited Reference

Claim 3 depends from Claim 2, and is considered to distinguish over the prior art for at least the same reasons given in support thereof. In addition, Claim 3 distinguishes over the art in reciting the feature that an application file is associated with the plug-in classes, and the plug-in class loader that is associated with the identified class loader is enabled to locate the given plug-in class by specifying a single configuration value in the application file. This feature is taught in Applicants' specification, such as at page 20, lines 24-28, but is neither disclosed nor suggested by *Susarla*, either at paragraph [0184] or elsewhere.

VIII. Claim 8 Distinguishes over Cited Reference

Claim 8 depends from Claim 2, and is considered to distinguish over the prior art for at least the same reasons given in support thereof. In addition, Claim 8 distinguishes over the art in reciting the feature that responsive to a first application class loading a first plug-in class, a target class loader is identified within the class loader hierarchy that loaded a target class; a plug-in class loader is identified that is provided for and delegates to the target class loader; and the first plug-in class is loaded using the plug-in class loader. *Susarla,* however, neither shows nor suggests this feature, either at paragraph [0141] or elsewhere. Paragraph [0141] is directed to using a class loader controller to replace a class loader with a new class loader, and then using the new class loader to load a changed class. Accordingly, this paragraph fails to show or suggest the recitation of Claim 8. For example, Claim 8 is directed to using a plug-in class loader to load the initial first plug-in class, and not a new class that has been changed.

IX. Claims 10 and 11 Distinguish over Cited Reference

Claim 10 depends from Claim 2, and is considered to distinguish over the prior art for at least the same reasons given in support thereof. In addition, Claim 10 distinguishes over the art in reciting the feature that a particular one of the application classes is loaded by a particular class loader of the class loader hierarchy, and first and second plug-in classes associated with respective first and second application classes respectively are each specified to use the class loader of the

particular application, and the first and second plug-in classes are both loaded by the plug-in class loader that is associated with and delegates to the particular class loader. This feature is recited in Applicants' specification, such as at page 21, line 23 – page 22, line 7. *Susarla,* however, neither shows nor suggests this feature, either at paragraph [0101] or elsewhere.

Claim 11 depends from Claim 10, and is considered to distinguish over the art for at least the same reasons given in support thereof. In addition, Claim 11 is considered to distinguish over the art in reciting the feature that the first plug-in class and the second plug-in class are able to share data. *Susarla,* however, neither shows nor suggests this feature, either at paragraph [0159] or elsewhere.

X. Remaining Claims Distinguish over the Cited References

Independent Claims 12 and 22 respectively incorporate subject matter similar to the patentable subject matter of Claim 2, and are each considered to distinguish over the art for at least the same reasons given in support thereof.

Claims 4-9 and 13-21 depend from independent Claims 2 and 12, respectively, and are each considered to distinguish over the art for at least the same reasons given in support thereof.

Claim 18 is additionally considered to distinguish over the art for the same reasons given in support for Claim 8.

XI. Conclusion

It is respectfully urged that the subject application is patentable over Susarla and is now in condition for allowance.

The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: April 24, 2008

Respectfully submitted,

/James O. Skarsten/

James O. Skarsten
Reg. No. 28,346
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Attorney for Applicants